Python 代码整理汇总-3

Pandas_1Foundations

DataFrame OUTLINE:

- Import data to DataFrame from dict & csv file
- Write into file
- Dataframe Operations: index, slicing, head, tail, info • Broadcasting
- Plot in DataFrame

Import data To DataFrame From Dictionary

```python
## Create a DataFrame from a dictionary
import pandas as pd

# Pre-defined lists: country [names], Drives_Right [dr], Cars per capital[cpc]
names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']
dr = [True, False, False, False, True, True, True]
cpc = [809, 731, 588, 18, 200, 70, 45]

my_dict = {'country':names, 'drives_right':dr, 'cars_per_cap':cpc}
cars = pd.DataFrame(my_dict) # 用字典建立df

print(cars)
```

```
         country  drives_right  cars_per_cap
0  United States          True           809
1      Australia         False           731
2          Japan         False           588
3          India         False            18
4         Russia          True           200
5        Morocco          True            70
6          Egypt          True            45
```

```python
## Optional: use zip
my_dict_zip = dict(zip(['country', 'drives_right', 'cars_per_cap'], [names, dr, cpc]))
cars_zip = pd.DataFrame(my_dict_zip)
print(cars_zip)
```

```
         country  drives_right  cars_per_cap
0  United States          True           809
1      Australia         False           731
2          Japan         False           588
3          India         False            18
4         Russia          True           200
5        Morocco          True            70
6          Egypt          True            45
```

```python
## Specify the row labels 自定义label（列也可以）
row_labels = ['US', 'AUS', 'JAP', 'IN', 'RU', 'MOR', 'EG']
cars.index = row_labels
print(cars)
```

```
           country  drives_right  cars_per_cap
US   United States          True           809
AUS      Australia         False           731
JAP          Japan         False           588
IN           India         False            18
RU          Russia          True           200
MOR        Morocco          True            70
EG           Egypt          True            45
```

Import data To DataFrame From CSV file

```python
## Create DataFrame from a CSV(Comma-Seperated Values) file
cars = pd.read_csv('cars.csv') # 分隔符默认为“,”
print(cars)
cars = pd.read_csv('cars.csv', index_col = 0) # 第一列为index
print(cars)
```

```
  Unnamed: 0  cars_per_cap        country  drives_right
0         US           809  United States          True
1        AUS           731      Australia         False
2        JAP           588          Japan         False
3         IN            18          India         False
4         RU           200         Russia          True
5        MOR            70        Morocco          True
6         EG            45          Egypt          True
     cars_per_cap        country  drives_right
US            809  United States          True
```

```
AUS          731      Australia         False
JAP          588          Japan         False
IN            18          India         False
RU           200         Russia          True
MOR           70        Morocco          True
EG            45          Egypt          True
```

other import keywords: Header, Names, na_value

```
 ## header = None
cars = pd.read_csv('cars.csv', header=None)
cars
```

	0	1	2	3
0	NaN	cars_per_cap	country	drives_right
1	US	809	United States	True
2	AUS	731	Australia	False
3	JAP	588	Japan	False
4	IN	18	India	False
5	RU	200	Russia	True
6	MOR	70	Morocco	True
7	EG	45	Egypt	True

```
## header = 0
cars = pd.read_csv('cars.csv', header=0) #第一行为header
cars
```

	Unnamed: 0	cars_per_cap	country	drives_right
0	US	809	United States	True
1	AUS	731	Australia	False
2	JAP	588	Japan	False
3	IN	18	India	False
4	RU	200	Russia	True
5	MOR	70	Morocco	True
6	EG	45	Egypt	True

```
col_names = ['A', 'B', 'C']
cars = pd.read_csv('cars.csv', names = col_names) # 设定列名
cars
```

	A	B	C
NaN	cars_per_cap	country	drives_right
US	809	United States	True
AUS	731	Australia	False
JAP	588	Japan	False
IN	18	India	False
RU	200	Russia	True
MOR	70	Morocco	True
EG	45	Egypt	True

```
col_names = ['A', 'B', 'C']
cars = pd.read_csv('cars.csv', names = col_names, na_values = {'A':['18','45']}) # A列中18, 45换成NaN
cars
```

	A	B	C
NaN	cars_per_cap	country	drives_right
US	809	United States	True
AUS	731	Australia	False
JAP	588	Japan	False
IN	NaN	India	False
RU	200	Russia	True
MOR	70	Morocco	True
EG	NaN	Egypt	True

Import Big Data File

```
def count_entries(csv_file, c_size, colname): # 不适合一次性读取
    """Return a dictionary with counts of
    occurrences as value for each key."""

    # Initialize an empty dictionary: counts_dict
    counts_dict = {}

    # Iterate over the file chunk by chunk
    for chunk in pd.read_csv(csv_file, chunksize=c_size):
        # Iterate over the column in DataFrame
        for entry in chunk[colname]: # 仅统计指定columns
            if entry in counts_dict.keys():
                counts_dict[entry] += 1
            else:
                counts_dict[entry] = 1
    # Return counts_dict
    return counts_dict

# Call count_entrieds(): result_counts
result_counts = count_entries('cars.csv', 10, 'drives_right')

# Print result_counts
print(result_counts)
```

```
{True: 4, False: 3}
```

Write into files

```
out_csv = 'newcsvFile.csv'
cars.to_csv(out_csv)
#cars.to_csv(out_csv, sep='\t')
#cars.to_excel('newexcelfile.xlsx')
```

DataFrame Operations: Index

```
cars = pd.read_csv('cars.csv', index_col = 0)
print(cars)

## show the type of dataframe
print(type(cars))
## show the shape
print(cars.shape)
## show the columns
print(cars.columns)
for item in cars.columns:
    print(item)
## show the index
```

```
print(cars.index)
for item in cars.index:
    print(item)
```

```
     cars_per_cap        country  drives_right
US            809  United States          True
AUS           731      Australia         False
JAP           588          Japan         False
IN             18          India         False
RU            200         Russia          True
MOR            70        Morocco          True
EG             45          Egypt          True
<class 'pandas.core.frame.DataFrame'>
(7, 3)
Index(['cars_per_cap', 'country', 'drives_right'], dtype='object')
cars_per_cap
country
drives_right
Index(['US', 'AUS', 'JAP', 'IN', 'RU', 'MOR', 'EG'], dtype='object')
US
AUS
JAP
IN
RU
MOR
EG
```

```
## Show the column labels
print("Column Labels: " + ", ".join(cars)) # 将序列中的元素以指定的字符连接生成一个新的字符串。
#for col in cars:
#    print(col)

## Show the row label and row content
for lab, row in cars.iterrows():
    print("-------Label: ", lab)
    print(row)
```

```
Column Labels: cars_per_cap, country, drives_right
-------Label:  US
cars_per_cap              809
country         United States
drives_right             True
Name: US, dtype: object
-------Label:  AUS
cars_per_cap          731
country         Australia
drives_right        False
Name: AUS, dtype: object
-------Label:  JAP
cars_per_cap      588
country         Japan
drives_right    False
Name: JAP, dtype: object
-------Label:  IN
cars_per_cap       18
country         India
drives_right    False
Name: IN, dtype: object
-------Label:  RU
cars_per_cap       200
country         Russia
drives_right      True
Name: RU, dtype: object
-------Label:  MOR
cars_per_cap         70
country         Morocco
drives_right       True
Name: MOR, dtype: object
-------Label:  EG
cars_per_cap       45
country         Egypt
drives_right     True
Name: EG, dtype: object
```

- for循环，短的循环放外面，长的放里面

- 调取dataframe时，存储是先行后列，for也要先行后列

DataFrame Operations: Slicing

```
cars
```

	cars_per_cap	country	drives_right
US	809	United States	True
AUS	731	Australia	False
JAP	588	Japan	False
IN	18	India	False
RU	200	Russia	True
MOR	70	Morocco	True
EG	45	Egypt	True

```
cars.iloc[:3,:]
#cars.iloc[-2:, :]
```

	cars_per_cap	country	drives_right
US	809	United States	True
AUS	731	Australia	False
JAP	588	Japan	False

```
cars.loc['US']
```

```
cars_per_cap              809
country         United States
drives_right             True
Name: US, dtype: object
```

```
cars.loc[:, 'cars_per_cap']
```

```
US     809
AUS    731
JAP    588
IN      18
RU     200
MOR     70
EG      45
Name: cars_per_cap, dtype: int64
```

```
cars.cars_per_cap
```

```
US     809
AUS    731
JAP    588
IN      18
RU     200
MOR     70
EG      45
Name: cars_per_cap, dtype: int64
```

```
print(cars['cars_per_cap'])
print(type(cars['cars_per_cap']))

cars['cars_per_cap'].values
```

```
US     809
AUS    731
JAP    588
IN      18
RU     200
MOR     70
EG      45
Name: cars_per_cap, dtype: int64
<class 'pandas.core.series.Series'>
```

```
array([809, 731, 588,  18, 200,  70,  45])
```

DataFrame Operations: Head, Tail, Info

```
cars
```

	cars_per_cap	country	drives_right
US	809	United States	True
AUS	731	Australia	False
JAP	588	Japan	False
IN	18	India	False
RU	200	Russia	True
MOR	70	Morocco	True
EG	45	Egypt	True

```
cars.head(3) ## Default: 5
```

	cars_per_cap	country	drives_right
US	809	United States	True
AUS	731	Australia	False
JAP	588	Japan	False

```
cars.tail() # 末尾
```

	cars_per_cap	country	drives_right
JAP	588	Japan	False
IN	18	India	False
RU	200	Russia	True
MOR	70	Morocco	True
EG	45	Egypt	True

```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7 entries, US to EG
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   cars_per_cap  7 non-null      int64
 1   country       7 non-null      object
 2   drives_right  7 non-null      bool
dtypes: bool(1), int64(1), object(1)
memory usage: 475.0+ bytes
```

```
cars.describe()
```

	cars_per_cap
count	7.000000
mean	351.571429
std	345.595552
min	18.000000
25%	57.500000
50%	200.000000
75%	659.500000
max	809.000000

DataFrame -> series -> ndarray

```
cpc = cars['cars_per_cap']
print(type(cpc))
print(cpc.head(4))

cpc_value = cpc.values
print(type(cpc_value))
print(cpc_value)
```

```
<class 'pandas.core.series.Series'>
US     809
AUS    731
JAP    588
IN      18
Name: cars_per_cap, dtype: int64
<class 'numpy.ndarray'>
[809 731 588  18 200  70  45]
```

DataFrame: Broadcasting

```
cars
```

	cars_per_cap	country	drives_right
US	809	United States	True
AUS	731	Australia	False
JAP	588	Japan	False
IN	18	India	False
RU	200	Russia	True
MOR	70	Morocco	True
EG	45	Egypt	True

```
import numpy as np
cars.iloc[:3,0] = np.nan
cars
```

	cars_per_cap	country	drives_right
US	NaN	United States	True
AUS	NaN	Australia	False
JAP	NaN	Japan	False
IN	18.0	India	False
RU	200.0	Russia	True
MOR	70.0	Morocco	True
EG	45.0	Egypt	True

```
cars['NewColumn'] = 0.0
cars
```

	cars_per_cap	country	drives_right	NewColumn
US	NaN	United States	True	0.0
AUS	NaN	Australia	False	0.0
JAP	NaN	Japan	False	0.0
IN	18.0	India	False	0.0
RU	200.0	Russia	True	0.0
MOR	70.0	Morocco	True	0.0
EG	45.0	Egypt	True	0.0

```
## add column in DataFrame
for lab, row in cars.iterrows():
    cars.loc[lab, "COUNTRY"] = row['country'].upper()
print(cars)

## More efficient way: using .apply
cars['COUNTRY'] = cars['country'].apply(str.upper) # 速度更快
print(cars)

## 处理数值类型的话，用np来处理更快
```

```
     cars_per_cap        country  drives_right  NewColumn        COUNTRY
US            NaN  United States          True        0.0  UNITED STATES
AUS           NaN      Australia         False        0.0      AUSTRALIA
JAP           NaN          Japan         False        0.0          JAPAN
IN           18.0          India         False        0.0          INDIA
RU          200.0         Russia          True        0.0         RUSSIA
MOR          70.0        Morocco          True        0.0        MOROCCO
EG           45.0          Egypt          True        0.0          EGYPT
     cars_per_cap        country  drives_right  NewColumn        COUNTRY
US            NaN  United States          True        0.0  UNITED STATES
AUS           NaN      Australia         False        0.0      AUSTRALIA
JAP           NaN          Japan         False        0.0          JAPAN
IN           18.0          India         False        0.0          INDIA
RU          200.0         Russia          True        0.0         RUSSIA
MOR          70.0        Morocco          True        0.0        MOROCCO
EG           45.0          Egypt          True        0.0          EGYPT
```

Series and DataFrame

```
## series
print(cars['cars_per_cap'])
print(type(cars['cars_per_cap']))

## DataFrame
print(cars[['cars_per_cap']])
print(type(cars[['cars_per_cap']]))
```

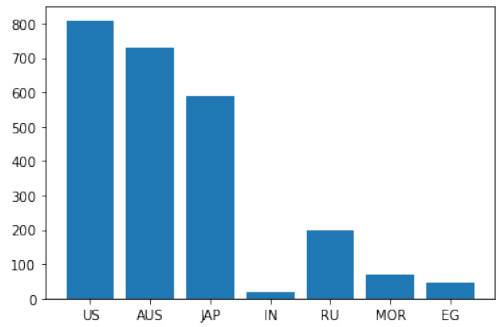
```
US       NaN
AUS      NaN
JAP      NaN
IN      18.0
RU     200.0
MOR     70.0
EG      45.0
Name: cars_per_cap, dtype: float64
<class 'pandas.core.series.Series'>
     cars_per_cap
US            NaN
AUS           NaN
JAP           NaN
IN           18.0
RU          200.0
MOR          70.0
EG           45.0
<class 'pandas.core.frame.DataFrame'>
```

Plot data in DataFrame

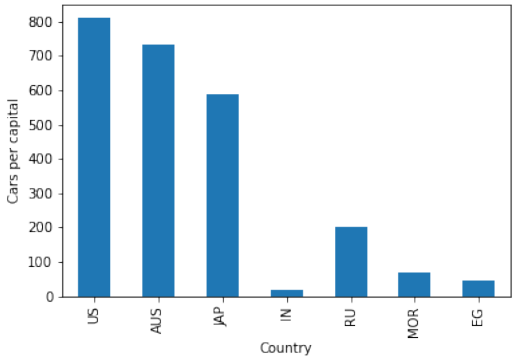
```
import matplotlib.pyplot as plt
cars = pd.read_csv('cars.csv', index_col = 0)
cars
```

	cars_per_cap	country	drives_right
US	809	United States	True
AUS	731	Australia	False
JAP	588	Japan	False
IN	18	India	False
RU	200	Russia	True
MOR	70	Morocco	True
EG	45	Egypt	True

```
## Plot using Matplotlib
cpc = cars['cars_per_cap'].values
xlabel = cars.index.values
ax = plt.bar(xlabel, cpc)
plt.savefig('cpc.pdf')
plt.show()
```



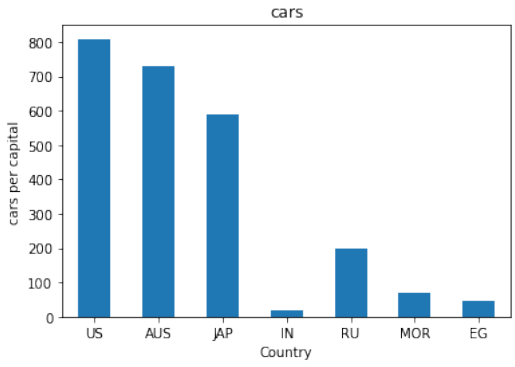
```
## plot using pandas
cpc = cars['cars_per_cap']
ax = cpc.plot.bar(rot = 90) #标签旋转90度
ax.set_xlabel('Country')
ax.set_ylabel('Cars per capital')
plt.show()
```




```
cars
```

	cars_per_cap	country	drives_right
US	809	United States	True
AUS	731	Australia	False
JAP	588	Japan	False
IN	18	India	False
RU	200	Russia	True
MOR	70	Morocco	True
EG	45	Egypt	True

```
ax = cars['cars_per_cap'].plot.bar(rot=0)
ax.set(xlabel='Country', ylabel='cars per capital', title = 'cars')
plt.show()
```



2_ExploratoryDataAnalysis

Example data set: **iris.csv**

iris:

- 150 observations,
- 4 features [Sepal Length, Sepal Width, Petal Length, Petal Width], • 3 species [setosa, versicolor,Virginica]

```
## import data
import pandas as pd
import matplotlib.pyplot as plt

iris = pd.read_csv('iris.csv')
col_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
iris.columns = col_names
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
print(iris.shape)
iris.info()
```

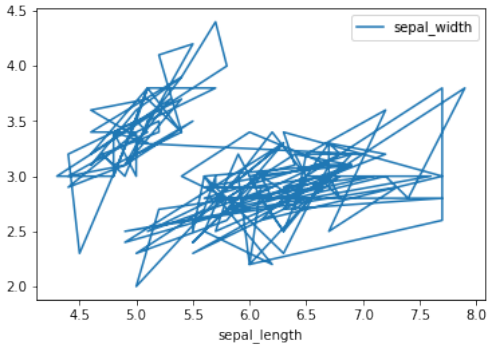
```
(150, 5)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Visual Exploratory Data Analysis

Line, Scatter, Box, Histogram

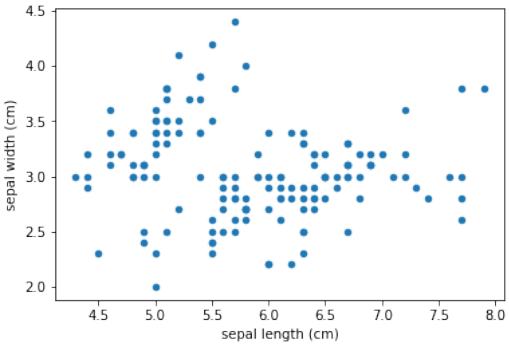
Line Plot

```
iris.plot(x='sepal_length', y = 'sepal_width')
plt.show()
```



Scatter Plot

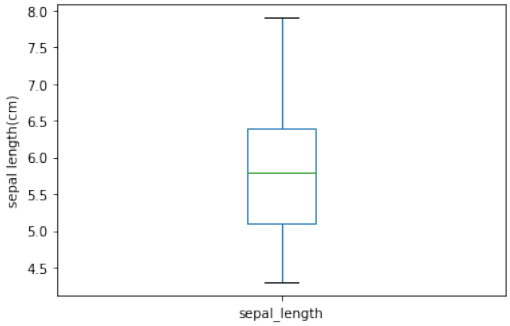
```
iris.plot(x='sepal_length', y = 'sepal_width',kind='scatter')
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.show()
```



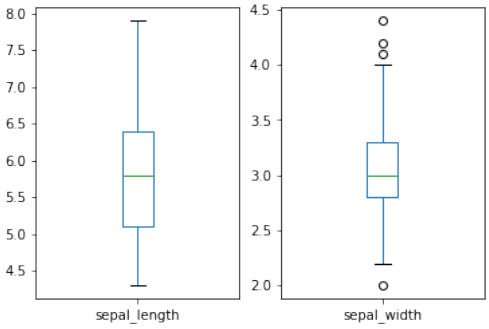
Box Plot

[Max, third quartile 75%, median, quartile 25%, Min]

```
iris.plot(y = 'sepal_length', kind='box')
plt.ylabel('sepal length(cm)')
plt.show()
```

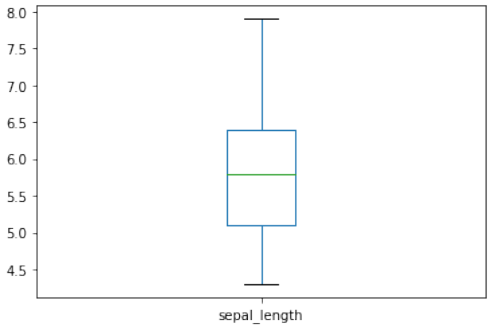



```
iris[['sepal_length', 'sepal_width']].plot(kind='box', subplots=True)
plt.show()
```



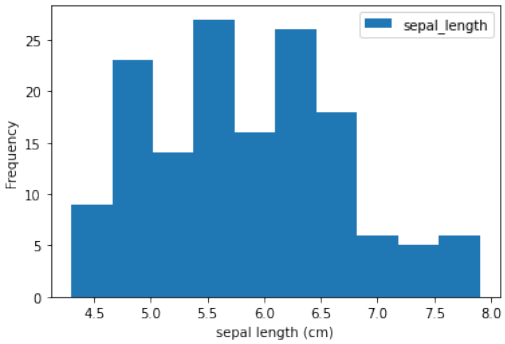

```
iris.sepal_length.plot(kind='box')
```

```
<AxesSubplot:>
```

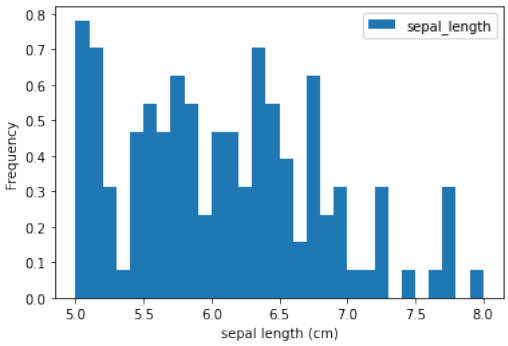


Histogram

```
iris.plot(y = 'sepal_length', kind='hist')
plt.xlabel('sepal length (cm)')
plt.show()
```



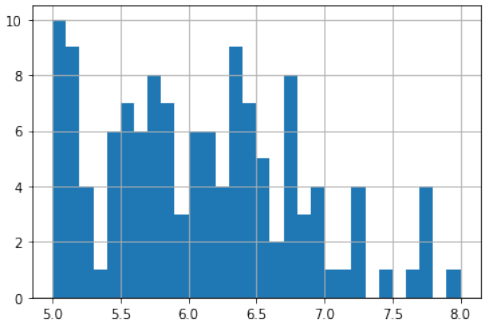

```
## Options for histogram
# bins: integer, number of bins
# range: tuple, (min, max)
# density: boolean, normalized to 1
# cumulative: boolean, Compute Cumulative Distribution Function

iris.plot(y = 'sepal_length', kind='hist', bins = 30, range=(5,8), density=True)
plt.xlabel('sepal length (cm)')
plt.show()
```



different plot methods

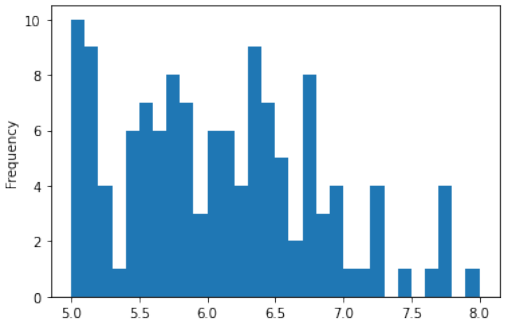
```
iris['sepal_length'].hist(bins = 30, range=(5,8))
```

```
<AxesSubplot:>
```



```
iris['sepal_length'].plot(bins = 30, range=(5,8), kind='hist')
```

```
<AxesSubplot:ylabel='Frequency'>
```



Statistical Exploratory Data Analysis

```
iris.tail(6)
```

	sepal_length	sepal_width	petal_length	petal_width	species
144	6.7	3.3	5.7	2.5	virginica
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Summary Statistics

```
iris.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Counts

```
iris['sepal_length'].count()
```

```
150
```

```
iris[['sepal_length', 'petal_length']].count()
```

```
sepal_length    150
petal_length    150
dtype: int64
```

Averages

```
iris['sepal_length'].mean()
```

```
5.843333333333335
```

```
iris.mean()
```

```
sepal_length    5.843333
sepal_width     3.054000
petal_length    3.758667
petal_width     1.198667
dtype: float64
```

Standard Deviations(std)

```
iris.std()
```

```
sepal_length    0.828066
sepal_width     0.433594
petal_length    1.764420
petal_width     0.763161
dtype: float64
```

Medians

```
iris.median()
```

```
sepal_length    5.80
sepal_width     3.00
petal_length    4.35
petal_width     1.30
dtype: float64
```

Quantiles

```
iris.quantile(0.5)
```

```
sepal_length    5.80
sepal_width     3.00
petal_length    4.35
petal_width     1.30
Name: 0.5, dtype: float64
```

Max, Min

```
print(iris['sepal_length'].max())
iris.min()
```

```
7.9
```

```
sepal_length       4.3
sepal_width        2.0
petal_length       1.0
petal_width        0.1
species         setosa
dtype: object
```

Unique

```
iris['species'].unique()
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

3_TimeSeriesPandas

Time Serise OUTLINE:

- import csv data with datatime • Resampling time series data
- Upsampling
- Visulization

datetime format

Import csv data with datetime

```
import pandas as pd

sales = pd.read_csv('sales.csv')
```

```
sales.head()
```

	Date	Company	Product	Units
0	2015-02-26 08:57:45	Streeplex	Service	4
1	2015-02-16 12:09:19	Hooli	Software	10
2	2015-02-03 14:14:18	Initech	Software	13
3	2015-02-02 08:33:01	Hooli	Software	3
4	2015-02-25 00:29:00	Initech	Service	10

```
sales = pd.read_csv('sales.csv', index_col = 'Date', parse_dates=True) # 转化成时间格式
sales.head()
```

	Company	Product	Units
Date			
2015-02-26 08:57:45	Streeplex	Service	4
2015-02-16 12:09:19	Hooli	Software	10
2015-02-03 14:14:18	Initech	Software	13
2015-02-02 08:33:01	Hooli	Software	3
2015-02-25 00:29:00	Initech	Service	10

```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 20 entries, 2015-02-26 08:57:45 to 2015-02-21 20:41:47
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Company  20 non-null     object
 1   Product  20 non-null     object
 2   Units    20 non-null     int64 
dtypes: int64(1), object(2)
memory usage: 640.0+ bytes
```

slicing times

- alternative formats:
  - sales.loc['Februray 5, 2015']
  - sales.loc['2015-Feb-5']
  - sales.loc['2015.2.5'] # 其他日期格式
- whole month:
  - sales.loc['2015-2'] #某一月
- whole year:
  - sales.loc['2015'] # 某一年
- slicing:
  - sales.loc['2015-2-16':'2015-2-20']

```
print(sales.loc['2015-02-26 08:57:45'])
print('-'*30)
print(sales.loc['2015-02-26 08:57:45','Company'])
print('-'*30)
print(sales.iloc[0,0])
print('-'*30)
print(sales.loc['2015-2-5']) # 只匹配Date
```

```
                     Company  Product  Units
Date
2015-02-26 08:57:45  Streeplex  Service      4
------------------------------
Date
2015-02-26 08:57:45    Streeplex
Name: Company, dtype: object
------------------------------
Streeplex
------------------------------
                      Company   Product  Units
Date
2015-02-05 01:53:06  Acme Coporation  Software     19
2015-02-05 22:05:03            Hooli   Service     10
```

```
sales.loc['2015-2-16':'2015-2-20']
```

	Company	Product	Units
Date			
2015-02-16 12:09:19	Hooli	Software	10
2015-02-19 16:02:58	Mediacore	Service	10
2015-02-19 10:59:33	Mediacore	Hardware	16

```
sales.loc['2015-2-4':'2015-2-5', 'Units']
```

```
Date
2015-02-05 01:53:06    19
2015-02-04 21:52:45    14
2015-02-05 22:05:03    10
2015-02-04 15:36:29    13
Name: Units, dtype: int64
```

Convert string to datetime

```
dateSTR = pd.to_datetime(['2015-02-11 20:03:08', '2015-02-11 21:23','2015-02-11 21:31'])
dateSTR
```

```
DatetimeIndex(['2015-02-11 20:03:08', '2015-02-11 21:23:00',
               '2015-02-11 21:31:00'],
              dtype='datetime64[ns]', freq=None)
```

Reindexing DataFrame, Filling missing values

Ref: pands.DataFrame.reindex

```
sales.head()
```

	Company	Product	Units
Date			
2015-02-26 08:57:45	Streeplex	Service	4
2015-02-16 12:09:19	Hooli	Software	10
2015-02-03 14:14:18	Initech	Software	13
2015-02-02 08:33:01	Hooli	Software	3
2015-02-25 00:29:00	Initech	Service	10

pandas中的reindex方法可以为series和dataframe添加或者删除索引。
方法：serise.reindex()、dataframe.reindex()
如果新添加的索引没有对应的值，则默认为nan。如果减少索引，就相当于一个切片操作。

```
sales.reindex(dateSTR) # 没有的就显示NaN
```

	Company	Product	Units
2015-02-11 20:03:08	Initech	Software	7.0
2015-02-11 21:23:00	NaN	NaN	NaN
2015-02-11 21:31:00	NaN	NaN	NaN

```
sales = sales.sort_index() # 排序
sales.head()
```

	Company	Product	Units
Date			
2015-02-02 08:33:01	Hooli	Software	3
2015-02-02 20:54:49	Mediacore	Hardware	9
2015-02-03 14:14:18	Initech	Software	13
2015-02-04 15:36:29	Streeplex	Software	13
2015-02-04 21:52:45	Acme Coporation	Hardware	14

```
sales.reindex(dateSTR, method = 'nearest') # 参考数据最相近的就用那个去填充，类似fillna
```

	Company	Product	Units
2015-02-11 20:03:08	Initech	Software	7
2015-02-11 21:23:00	Initech	Software	7
2015-02-11 21:31:00	Hooli	Software	4

```
sales.reindex(dateSTR, method = 'backfill') #用后面的去填充
```

	Company	Product	Units
2015-02-11 20:03:08	Initech	Software	7
2015-02-11 21:23:00	Hooli	Software	4
2015-02-11 21:31:00	Hooli	Software	4

Resampling

- Statistical methods over different time intervals
  - mean(), sum(), count()
- Resampling Frequencies
  - 'min' or 'T' for minute, 'H', 'D', 'B' for business day, 'W', 'M', 'Q' for quarter, 'A' for year

  是对原样本重新处理的一个方法，是一个对常规时间序列数据重新采样和频率转换的便捷的方法。

```
sales.resample('D').mean() # D: daysales.head()
```

	Company	Product	Units
Date			
2015-02-02 08:33:01	Hooli	Software	3
2015-02-02 20:54:49	Mediacore	Hardware	9
2015-02-03 14:14:18	Initech	Software	13
2015-02-04 15:36:29	Streeplex	Software	13
2015-02-04 21:52:45	Acme Coporation	Hardware	14

```
sales.resample('d').mean().max()
```

```
Units    14.5
dtype: float64
```

```
sales.resample('w').count() # week
```

	Company	Product	Units
Date			
2015-02-08	8	8	8
2015-02-15	4	4	4
2015-02-22	5	5	5
2015-03-01	3	3	3

complex case

```
sales.loc[:, 'Units'].resample('2W').sum()
```

```
Date
2015-02-08    82
2015-02-22    79
2015-03-08    15
Freq: 2W-SUN, Name: Units, dtype: int64
```

Manipulating time series data

```
sales = pd.read_csv('sales.csv', parse_dates = ['Date']) # 将csv中的时间字符串转换成日期格式
sales.head()
```

	Date	Company	Product	Units
0	2015-02-26 08:57:45	Streeplex	Service	4
1	2015-02-16 12:09:19	Hooli	Software	10
2	2015-02-03 14:14:18	Initech	Software	13
3	2015-02-02 08:33:01	Hooli	Software	3
4	2015-02-25 00:29:00	Initech	Service	10

String methods

```
sales.Company.str.upper() # 首字母大写
sales.Company.head()
```

```
0    Streeplex
1        Hooli
2      Initech
3        Hooli
4      Initech
Name: Company, dtype: object
```

```
## Substring matching
sales['Product'].str.contains('ware').head()
```

```
0    False
1     True
2     True
3     True
4    False
Name: Product, dtype: bool
```

```
sales.Product[sales['Product'].str.contains('ware')].head()
```

```
1    Software
2    Software
3    Software
5    Software
7    Software
Name: Product, dtype: object
```

```
sales.Product.str.contains('ware').sum()
```

```
14
```

Datetime methods

```
## daytime: year:month:day:hour:minute:second
sales['Date'].dt.day.head() # 提取出“日”
```

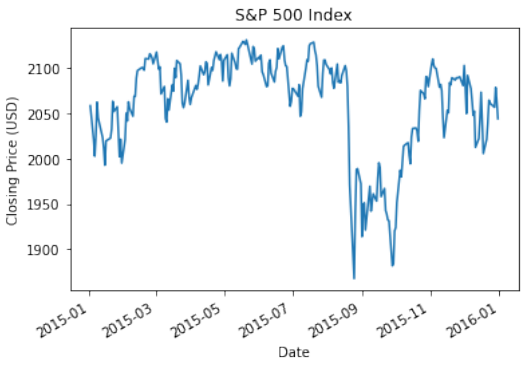
```
0    26
1    16
2     3
3     2
4    25
Name: Date, dtype: int64
```

Visualization

```
import matplotlib.pyplot as pltsp500 = pd.read_csv('sp500.csv', parse_dates = True, index_col = 0) sp500.head()
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2015-01-02	2058.899902	2072.360107	2046.040039	2058.199951	2708700000	2058.199951
2015-01-05	2054.439941	2054.439941	2017.339966	2020.579956	3799120000	2020.579956
2015-01-06	2022.150024	2030.250000	1992.439941	2002.609985	4460110000	2002.609985
2015-01-07	2005.550049	2029.609985	2005.550049	2025.900024	3805480000	2025.900024
2015-01-08	2030.609985	2064.080078	2030.609985	2062.139893	3934010000	2062.139893

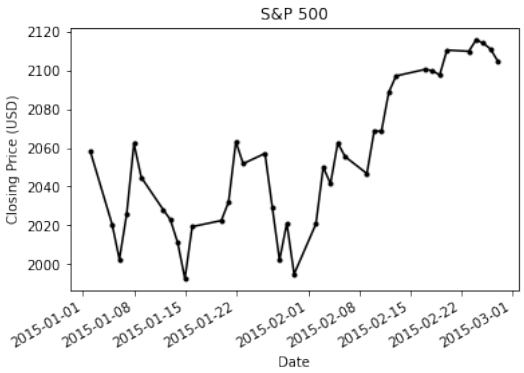
```
## line plot
sp500['Close'].plot(title = 'S&P 500 Index')
plt.ylabel('Closing Price (USD)')
plt.show()
```



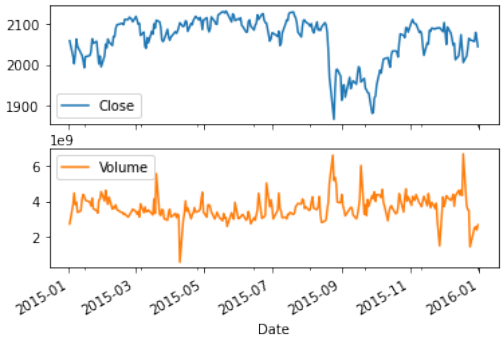

```
sp500.loc['2015-01-01':'2015-03-01', 'Close'].count()
```

```
39
```

```
sp500.loc['2015-01-01':'2015-03-01', 'Close'].plot(title = 'S&P 500', style ='k.-')
plt.ylabel('Closing Price (USD)')
plt.show()
```




```
## subplots
sp500.loc['2015', ['Close', 'Volume']].plot(subplots = True)
plt.show()
```



Pandas_4ClimateCase

Import data file

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

df = pd.read_csv('2011_climate.txt', header=None)
df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	34	35	36	37	38	39	40	41
0	13904	20110101	53	12	OVC045		10.00				...			29.95		AA			
1	13904	20110101	153	12	OVC049		10.00				...			30.01		AA			
2	13904	20110101	253	12	OVC060		10.00				...	030		30.01		AA			
3	13904	20110101	353	12	OVC065		10.00				...			30.03		AA			
4	13904	20110101	453	12	BKN070		10.00				...			30.04		AA			

5 rows × 44 columns

```
# df.info()
# df.columns
```

Add Columns

```
## Add columns
columns_label
='Wban,date,Time,StationType,sky_condition,sky_conditionFlag,visibility,visibilityFlag,wx_and_obst_to_vision,wx_and_obst_to_visionFlag,dry_bulb_faren,dry_bulb_farenFlag,dry_bulb_cel,dry_bulb_celFlag,wet_bulb_faren,wet_bulb_farenFlag,wet_bulb_cel,wet_bulb_celFlag,dew_point_faren,dew_point_farenFlag,dew_point_cel,dew_point_celFlag,relative_humidity,relative_humidityFlag,wind_speed,wind_speedFlag,wind_direction,wind_directionFlag,value_for_wind_character,value_for_wind_characterFlag,station_pressure,station_pressureFlag,pressure_tendency,pressure_tendencyFlag,presschange,presschangeFlag,sea_level_pressure,sea_level_pressureFlag,record_type,hourly_precip,hourly_precipFlag,altimeter,altimeterFlag,junk'
columns_list = columns_label.split(',')
df.columns = columns_list
df.head()
```

	Wban	date	Time	StationType	sky_condition	sky_conditionFlag	visibility	visibilityFlag	wx_and_obst_to_visior
0	13904	20110101	53	12	OVC045		10.00		
1	13904	20110101	153	12	OVC049		10.00		
2	13904	20110101	253	12	OVC060		10.00		
3	13904	20110101	353	12	OVC065		10.00		
4	13904	20110101	453	12	BKN070		10.00		

5 rows × 44 columns

```
## Remove some columns
columns_drop
='sky_conditionFlag,visibilityFlag,wx_and_obst_to_vision,wx_and_obst_to_visionFlag,dry_bulb_farenFlag,dry_bulb_celFlag,wet_bulb_farenFlag,wet_bulb_celFlag,dew_point_farenFlag,dew_point_celFlag,relative_humidityFlag,wind_speedFlag,wind_directionFlag,value_for_wind_character,value_for_wind_characterFlag,station_pressureFlag,pressure_tendencyFlag,pressure_tendency,presschange,presschangeFlag,sea_level_pressureFlag,hourly_precip,hourly_precipFlag,altimeter,record_type,altimeterFlag,junk'
columns_drop_list = columns_drop.split(',')
df_dropped = df.drop(columns_drop_list, axis = 'columns')
#df.head()
df_dropped.head()
```

	Wban	date	Time	StationType	sky_condition	visibility	dry_bulb_faren	dry_bulb_cel	wet_bulb_faren	wet_b
0	13904	20110101	53	12	OVC045	10.00	51	10.6	38	3.1
1	13904	20110101	153	12	OVC049	10.00	51	10.6	37	3.0
2	13904	20110101	253	12	OVC060	10.00	51	10.6	37	2.9
3	13904	20110101	353	12	OVC065	10.00	50	10.0	38	3.1
4	13904	20110101	453	12	BKN070	10.00	50	10.0	37	2.8

```
## OPTIONAL for columns name
with open('2011_climate_label.txt', 'r') as fid:
    s = fid.read().split('\n')
    columns_label = s[1]
    columns_drop = s[4]
    print(columns_label, '\n', columns_drop)
```

```
Wban,date,Time,StationType,sky_condition,sky_conditionFlag,visibility,visibilityFlag,wx_and_obst_to_vision,wx_and_obst_to_visionFlag,dry_bulb_faren,dry_bulb_farenFlag,dry_bulb_cel,dry_bulb_celFlag,wet_bulb_faren,wet_bulb_farenFlag,wet_bulb_cel,wet_bulb_celFlag,dew_point_faren,dew_point_farenFlag,dew_point_cel,dew_point_celFlag,relative_humidity,relative_humidityFlag,wind_speed,wind_speedFlag,wind_direction,wind_directionFlag,value_for_wind_character,value_for_wind_characterFlag,station_pressure,station_pressureFlag,pressure_tendency,pressure_tendencyFlag,presschange,presschangeFlag,sea_level_pressure,sea_level_pressureFlag,record_type,hourly_precip,hourly_precipFlag,altimeter,altimeterFlag,junk
sky_conditionFlag,visibilityFlag,wx_and_obst_to_vision,wx_and_obst_to_visionFlag,dry_bulb_farenFlag,dry_bulb_celFlag,wet_bulb_farenFlag,wet_bulb_celFlag,dew_point_farenFlag,dew_point_celFlag,relative_humidityFlag,wind_speedFlag,wind_directionFlag,value_for_wind_character,value_for_wind_characterFlag,station_pressureFlag,pressure_tendencyFlag,pressure_tendency,presschange,presschangeFlag,sea_level_pressureFlag,hourly_precip,hourly_precipFlag,altimeter,record_type,altimeterFlag,junk
```

Set Datatime as Index

REF:
Pandas.DataFrame.astype
Pandas.to_datetime

```
df_dropped.date
```

```
0        20110101
1        20110101
2        20110101
3        20110101
4        20110101
...
10332    20111231
10333    20111231
10334    20111231
10335    20111231
10336    20111231
Name: date, Length: 10337, dtype: int64
```

```
## Convert columns to pandas datetime object
# print(df_dropped.info())
# print(df_dropped.head())
## convert dtypes from int64 to str(object)
df_dropped['date'] = df_dropped['date'].astype('str')
```

```
## convert time -> hour+minute: 453 -> 0453 | 53 -> 0053
df_dropped['Time'] = df_dropped['Time'].apply(lambda x: '{:0>4}'.format(x))

## pandas datetime object
date_times = pd.to_datetime(df_dropped['date'] + df_dropped['Time'], format ='%Y%m%d%H%M')
## set_index
df_clean = df_dropped.set_index(date_times)
df_clean.head()
```

	Wban	date	Time	StationType	sky_condition	visibility	dry_bulb_faren	dry_bulb_cel	wet_bulb_faren
2011-01-01 00:53:00	13904	20110101	0053	12	OVC045	10.00	51	10.6	38
2011-01-01 01:53:00	13904	20110101	0153	12	OVC049	10.00	51	10.6	37
2011-01-01 02:53:00	13904	20110101	0253	12	OVC060	10.00	51	10.6	37
2011-01-01 03:53:00	13904	20110101	0353	12	OVC065	10.00	50	10.0	38
2011-01-01 04:53:00	13904	20110101	0453	12	BKN070	10.00	50	10.0	37

```
df_dropped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10337 entries, 0 to 10336
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Wban                10337 non-null  int64 
 1   date                10337 non-null  object
 2   Time                10337 non-null  object
 3   StationType         10337 non-null  int64 
 4   sky_condition       10337 non-null  object
 5   visibility          10325 non-null  object
 6   dry_bulb_faren      10337 non-null  object
 7   dry_bulb_cel        10337 non-null  object
 8   wet_bulb_faren      10337 non-null  object
 9   wet_bulb_cel        10337 non-null  object
 10  dew_point_faren     10337 non-null  object
 11  dew_point_cel       10337 non-null  object
 12  relative_humidity   10337 non-null  object
 13  wind_speed          10337 non-null  object
 14  wind_direction      10337 non-null  object
 15  station_pressure    10337 non-null  object
 16  sea_level_pressure  10337 non-null  objectd
types: int64(2), object(15)memory usage: 1.3+ MB
```

Clean numeric columns

```
## Read 'dry_bulb_faren' Temperature
df_clean.loc['2011.6.20 8AM':'2011.6.20 9AM', 'dry_bulb_faren']
```

```
2011-06-20 08:27:00     M
2011-06-20 08:28:00     M
2011-06-20 08:29:00     M
2011-06-20 08:30:00     M
2011-06-20 08:31:00     M
2011-06-20 08:32:00     M
2011-06-20 08:33:00     M
2011-06-20 08:34:00     M
2011-06-20 08:35:00     M
2011-06-20 08:53:00    83
2011-06-20 09:08:00    84
2011-06-20 09:53:00    88
Name: dry_bulb_faren, dtype: object
```

```
df_clean['dry_bulb_faren'] = pd.to_numeric(df_clean['dry_bulb_faren'], errors ='coerce') #coerce: set 'M' to NaN
df_clean.loc['2011.6.20 8AM':'2011.6.20 9AM', 'dry_bulb_faren']
```

```
2011-06-20 08:27:00     NaN
2011-06-20 08:28:00     NaN
2011-06-20 08:29:00     NaN
2011-06-20 08:30:00     NaN
2011-06-20 08:31:00     NaN
2011-06-20 08:32:00     NaN
2011-06-20 08:33:00     NaN
2011-06-20 08:34:00     NaN
2011-06-20 08:35:00     NaN
2011-06-20 08:53:00    83.0
2011-06-20 09:08:00    84.0
2011-06-20 09:53:00    88.0
Name: dry_bulb_faren, dtype: float64
```

```
df_clean['wind_speed'] = pd.to_numeric(df_clean['wind_speed'], errors='coerce')df_clean['visibility'] =
pd.to_numeric(df_clean['visibility'], errors='coerce')df_clean['dew_point_faren'] =
pd.to_numeric(df_clean['dew_point_faren'], errors='coerce')
df_clean.head()
```

	Wban	date	Time	StationType	sky_condition	visibility	dry_bulb_faren	dry_bulb_cel	wet_bulb_faren
2011-01-01 00:53:00	13904	20110101	0053	12	OVC045	10.0	51.0	10.6	38
2011-01-01 01:53:00	13904	20110101	0153	12	OVC049	10.0	51.0	10.6	37
2011-01-01 02:53:00	13904	20110101	0253	12	OVC060	10.0	51.0	10.6	37
2011-01-01 03:53:00	13904	20110101	0353	12	OVC065	10.0	50.0	10.0	38
2011-01-01 04:53:00	13904	20110101	0453	12	BKN070	10.0	50.0	10.0	37

Data Analysis

Statistical Features of Time Series : mean, median, max, min, sum, count

```
print(df_clean['dry_bulb_faren'].median())
print(df_clean.loc['2011-Apr':'2011-Jun','dry_bulb_faren'].max())
print(df_clean.loc['2011-Jan', 'dry_bulb_faren'].count())
```

```
72.0
104.0
907
```

Resample

```
daily_mean_2011 = df_clean.resample('D').mean()
daily_mean_2011
```

	Wban	StationType	visibility	dry_bulb_faren	dew_point_faren	wind_speed
2011-01-01	13904	12	10.000000	50.166667	20.500000	11.083333
2011-01-02	13904	12	10.000000	39.416667	19.708333	4.166667
2011-01-03	13904	12	10.000000	46.846154	35.500000	2.653846
2011-01-04	13904	12	5.071429	53.367347	50.408163	2.510204
2011-01-05	13904	12	7.672414	57.965517	40.068966	4.689655
...	...	...	...	...	...	...
2011-12-27	13904	12	10.000000	44.833333	32.125000	4.458333
2011-12-28	13904	12	10.000000	45.750000	35.166667	5.375000
2011-12-29	13904	12	10.000000	50.320000	35.600000	4.480000
2011-12-30	13904	12	10.000000	52.541667	37.875000	5.500000
2011-12-31	13904	12	10.000000	54.458333	42.416667	5.416667

365 rows × 6 columns

```
# dataframe -> series -> numpy array, NOTICE: values is attribute, not method
daily_temperature_2011 = daily_mean_2011['dry_bulb_faren'].values
print(type(daily_mean_2011['dry_bulb_faren']))
print(type(daily_temperature_2011))
```

```
<class 'pandas.core.series.Series'>
<class 'numpy.ndarray'>
```

Practise:

- import '2010_climate.csv' and format it
- set 'Date' column as its index
- Compare the temperature for 2011 and 2010

```
## add another dataset: weather_data_2010.csv
df_climate = pd.read_csv('2010_climate.csv', header = 0)
print(df_climate.head())
df_climate2 = df_climate.set_index('Date')
print(df_climate2.head())
df_climate3 = df_climate2.reset_index()
print(df_climate3.head())
```

```
   Temperature  DewPoint  Pressure            Date
0         46.2      37.5       1.0  20100101 00:00
1         44.6      37.1       1.0  20100101 01:00
2         44.1      36.9       1.0  20100101 02:00
3         43.8      36.9       1.0  20100101 03:00
4         43.5      36.8       1.0  20100101 04:00
                 Temperature  DewPoint  Pressure
Date
20100101 00:00          46.2      37.5       1.0
20100101 01:00          44.6      37.1       1.0
20100101 02:00          44.1      36.9       1.0
20100101 03:00          43.8      36.9       1.0
20100101 04:00          43.5      36.8       1.0
              Date  Temperature  DewPoint  Pressure
0  20100101 00:00          46.2      37.5       1.0
1  20100101 01:00          44.6      37.1       1.0
2  20100101 02:00          44.1      36.9       1.0
3  20100101 03:00          43.8      36.9       1.0
4  20100101 04:00          43.5      36.8       1.0
```

```
df_climate['Date'] = pd.to_datetime(df_climate['Date'], format = '%Y%m%d %H:%M', errors='coerce')
df_climate = df_climate.set_index('Date')
df_climate.head()
```

	Temperature	DewPoint	Pressure
Date			
2010-01-01 00:00:00	46.2	37.5	1.0
2010-01-01 01:00:00	44.6	37.1	1.0
2010-01-01 02:00:00	44.1	36.9	1.0
2010-01-01 03:00:00	43.8	36.9	1.0
2010-01-01 04:00:00	43.5	36.8	1.0

```
daily_climate = df_climate.resample('D').mean()
daily_climate

# daily_temperature_climate = daily_climate.reset_index()
# daily_temperature_climate.head()
```

	Temperature	DewPoint	Pressure
Date			
2010-01-01	49.337500	37.716667	1.0
2010-01-02	49.795833	38.370833	1.0
2010-01-03	49.900000	38.279167	1.0
2010-01-04	49.729167	38.008333	1.0
2010-01-05	49.841667	38.087500	1.0
...	...	...	...
2010-12-27	49.204167	37.816667	1.0
2010-12-28	48.979167	37.329167	1.0
2010-12-29	48.804167	37.025000	1.0
2010-12-30	49.008333	37.325000	1.0
2010-12-31	49.195833	37.450000	1.0

365 rows × 3 columns

```
daily_temperature_2010 = daily_climate['Temperature']
print(np.mean(daily_temperature_2011 - daily_temperature_2010))
```

```
1.3301831870056482
```

Compare Sunny and Overcast

```
is_sky_clear = df_clean['sky_condition'] == 'CLR'
sunny = df_clean[is_sky_clear]
sunny.head()
```

	Wban	date	Time	StationType	sky_condition	visibility	dry_bulb_faren	dry_bulb_cel	wet_bulb_faren
2011-01-01 13:53:00	13904	20110101	1353	12	CLR	10.0	59.0	15.0	45
2011-01-01 14:53:00	13904	20110101	1453	12	CLR	10.0	59.0	15.0	45
2011-01-01 15:53:00	13904	20110101	1553	12	CLR	10.0	57.0	13.9	44
2011-01-01 16:53:00	13904	20110101	1653	12	CLR	10.0	55.0	12.8	43
2011-01-01 17:53:00	13904	20110101	1753	12	CLR	10.0	50.0	10.0	40

```
sunny_daily_max = sunny.resample('D').max()
sunny_daily_max.head()
```

	Wban	date	Time	StationType	sky_condition	visibility	dry_bulb_faren	dry_bulb_cel	wet_bulb_faren
2011-01-01	13904.0	20110101	2353	12.0	CLR	10.0	59.0	8.3	45
2011-01-02	13904.0	20110102	2253	12.0	CLR	10.0	35.0	1.7	32
2011-01-03	13904.0	20110103	0453	12.0	CLR	10.0	32.0	0.0	29
2011-01-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-01-05	13904.0	20110105	2353	12.0	CLR	10.0	35.0	1.7	33

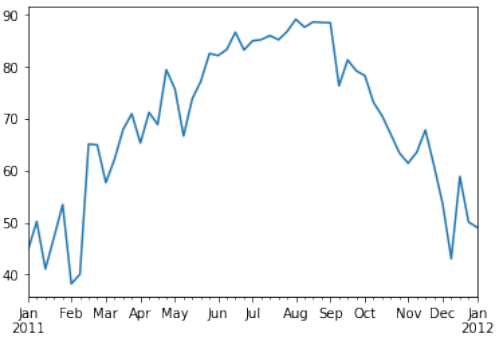
```
is_sky_overcast = df_clean['sky_condition'].str.contains('OVC')
overcast = df_clean.loc[is_sky_overcast]
overcast_daily_max = overcast.resample('D').max()
```

```
# Calculate the mean of sunny_daily_max
sunny_daily_max_mean = sunny_daily_max.mean()

# Calculate the mean of overcast_daily_max
overcast_daily_max_mean = overcast_daily_max.mean()

# Print the difference (sunny minus overcast)
print(sunny_daily_max_mean - overcast_daily_max_mean)
```

```
Wban              0.000000
StationType       0.000000
visibility        0.174359
dry_bulb_faren    6.504304
dew_point_faren  -4.339286
wind_speed       -3.246062
dtype: float64
```

Visualization

```
## Show the visibility and dry_bulb_faren in figure
weekly_mean = df_clean['dry_bulb_faren'].resample('W').mean()
weekly_mean.plot()
plt.show()
```




```
monthly_max = df_clean[['dew_point_faren', 'dry_bulb_faren']].resample('M').max()
monthly_max.plot(kind='hist', bins=8, alpha=0.5, subplots=True)
plt.show()
```